## Contents

1. What is FIQL?  
2. How does FIQL work?  
3. Installing *fiql_parser*  
4. Using *fiql_parser*  
5. API Reference  
   5.1. FIQL Parser API  
   5.1.1. Constants  
   5.1.2. Exceptions  
   5.1.3. Operator  
   5.1.4. Constraint  
   5.1.5. Expression  
   5.1.6. Parser  

Python Module Index
release 0.15

A Python parser for the Feed Item Query Language (FIQL).
What is FIQL?

From the FIQL draft [https://tools.ietf.org/html/draft-nottingham-atompub-fiql-00]:

The Feed Item Query Language (FIQL, pronounced “fickle”) is a simple but flexible, URI-friendly syntax for expressing filters across the entries in a syndicated feed.
How does FIQL work?

A Feed Item Query string looks something like this:

```
last_name==foo*, (age.lt=55; age.gt=5)
```

The above query string is looking for all records with `last_name` starting with “foo” OR `age` less than 55 AND greater than 5. The parentheses in the query work the same as they do in any logical expression.
Chapter 2. How does FIQL work?
Installing fiql_parser

From PyPi

$ pip install fiql-parser

From source (tar.gz or checkout)

Unpack the archive, enter the fiql_parser directory and run:

$ python setup.py install
Currently the functionality is pretty limited so there isn’t a lot to say on how to use it.

### 4.1 Parsing a FIQL formatted string

```python
from fiql_parser import parse_str_to_expression

fiql_str = "last_name==foo*,(age=lt=55;age=gt=5)"
expression = parse_str_to_expression(fiql_str)

# to_python()’s output changed with Version 0.11.
print expression.to_python()
# Output of above would be:
['OR',
 ('last_name', '==', 'foo*'),
 'AND',
 ('age', '<', '55'),
 ('age', '>', '5')
]
```

### 4.2 Building an Expression

**Method One**

```python
from fiql_parser import (Expression, Constraint, Operator)

expression = Expression()
expression.add_element(Constraint('last_name', '==', 'foo*'))
expression.add_element(Operator(','))
sub_expression = Expression()
sub_expression.add_element(Constraint('age', '=lt=', '55'))
sub_expression.add_element(Operator(';'))
sub_expression.add_element(Constraint('age', '=gt=', '5'))
expression.add_element(sub_expression)

print str(expression)
# Output of above would be:
"last_name==foo*,(age=lt=55;age=gt=5)"
```
Method Two (Changed in Version 0.11)

```python
from fiql_parser import Constraint

expression = Constraint('last_name', '==', 'foo*').op_or(
    Constraint('age', '=lt=', '55').op_and(
        Constraint('age', '=gt=', '5')
    )
)

fiql_str = str(expression)
# Output of above would be:
"last_name==foo*,(age=lt=55;age=gt=5)"
```

- genindex
- modindex
- search
CHAPTER 5

API Reference

Information on specific functions, classes, methods, and exception.

5.1 FIQL Parser API

© Copyright 2014-2015, by Serge Domkowski.

Note: This code includes a few modifications to rules in the FIQL draft.

The rule defined for Comparison has been modified to deal with an inconsistency in the draft documentation. The change fixes an issue where the string “==” was NOT a valid Comparison and thus made most of the examples in the FIQL draft incorrect.

The accepted arg chars to have been modified to include “:”. This change fixes the issue where RFC 3339 compliant DateTime values were not valid unless the “:” was percent-encoded. This contradicted the FIQL draft date_str examples. Since “:” is a valid character in an HTTP query *( pchar / "/" / ":" ), I opted to fix the issue by simply allowing the “:” in addition to the other arg chars.

5.1.1 Constants

Compiled and uncompiled regular expressions representing the various syntax rules used in the FIQL specification.

fiql_parser.constants.PCT_ENCODING_REGEX
Regular expression representing Percent-Encoding (RFC 3986#section-2.1).

fiql_parser.constants.UNRESERVED_REGEX
Regular expression representing Unreserved Characters (RFC 3986#section-2.3).

fiql_parser.constants.FIQL_DELIM_REGEX
Regular expression representing the FIQL Delimiter (FIQL Draft#section-3.2).

fiql_parser.constants.COMPARISON_REGEX
Regular expression representing the FIQL Comparison operator; e.g., “=gt=” (FIQL Draft#section-3.2). This rule includes a modification to the rule in the FIQL draft that correctly allows for a string with no ALPHA characters.

fiql_parser.constants_SELECTOR_REGEX
Regular expression representing the FIQL Selector (FIQL Draft#section-3.2). The Selector identifies the portion of an entry that a Constraint applies to.
fiql_parser.constants.ARG_CHAR_REGEX
Regular expression representing the characters allowed in a FIQL Argument (FIQL Draft#section-3.2). This rule includes a modification to the rule in the FIQL draft that allows for ":" in arguments (Example: “2015-08-27T10:30:00Z”).

fiql_parser.constants.ARGUMENT_REGEX
Regular expression representing the FIQL Argument (FIQL Draft#section-3.2). The Argument identifies the value that the Comparison operator should use when validating the Constraint.

fiql_parser.constants.CONSTRAINT_REGEX
Regular expression representing the FIQL Constraint (FIQL Draft#section-3.2). The Constraint, when processed, yields a boolean value.

fiql_parser.constants.CONSTRAINT_COMP
Compiled version of CONSTRAINT_REGEX.

fiql_parser.constants.COMPARISON_COMP
Compiled version of CONSTRAINT_REGEX as a full string.

5.1.2 Exceptions

The code in this package is intended to be used in one of two ways; Building the object representation of a FIQL expression directly, or building the object representation of a FIQL expression by parsing it from a FIQL string.

The Exception classes contained in this module are intended to provide the flexibility to differentiate between errors resulting from attempting to construct the expression object and those resulting from incorrectly formatted FIQL strings.

exception fiql_parser.exceptions.FiqlException
   Bases: exceptions.Exception
   Base Exception class for FIQL errors.

exception fiql_parser.exceptions.FiqlFormatException
   Bases: fiql_parser.exceptions.FiqlException
   Exception class for FIQL string parsing errors.

exception fiql_parser.exceptions.FiqlObjectException
   Bases: fiql_parser.exceptions.FiqlException
   Exception class for FIQL expression object errors.

5.1.3 Operator

FIQL has two operators. ";" which is the logical AND and "," for the logical OR where AND has a logical precedence which is higher than that of OR.

The operator module includes the code used for managing comparison operator acceptance, precedence, and representation of the FIQL Operator.

fiql_parser.operatorOPERATOR_MAP
dict of tuple
   Mappings of FIQL operators to common terms and their associated precedence.

class fiql_parser.operator.Operator (fiql_op_str)
   Bases: object
   The comparison Operator is the representation of the FIQL comparison operator.
value
    string
    The FIQL operator.

__cmp__(other)
    Compare using operator precedence.
    Parameters
    other (Operator) – The Operator we are comparing precedence against.
    Returns
    1 if greater than other, -1 if less than other, and 0 if of equal precedence of other.
    Return type
    integer

__eq__(other)
    Of equal precedence.
    Parameters
    other (Operator) – The Operator we are comparing precedence against.
    Returns
    True if of equal precedence of other.
    Return type
    boolean

__lt__(other)
    Of less than precedence.
    Parameters
    other (Operator) – The Operator we are comparing precedence against.
    Returns
    True if of less than precedence of other.
    Return type
    boolean

__str__()
    Represent the Operator instance as a string.
    Returns
    The represented Operator.
    Return type
    string

to_python()
    Deconstruct the Operator instance to a string.
    Returns
    The deconstructed Operator.
    Return type
    string

5.1.4 Constraint

The FIQL Constraint is the building block of the FIQL Expression. A FIQL Constraint is, on its own, a very simple Expression.

The constraint module includes the code used for managing comparison acceptance and representation of the FIQL Constraint.

fiql_parser.constraint.COMPARISON_MAP
dict
    Mappings for common FIQL comparisons.

class fiql_parser.constraint.Constraint (selector, comparison=None, argument=None)
    Bases: fiql_parser.expression.BaseExpression
    The Constraint is the smallest logical unit for a FIQL Expression. It itself must evaluate to True or False and contains no smaller unit which itself can evaluate to True or False.
**selector**

*string*

Constraint selector.

**comparison**

*string*

Constraint comparison operator.

**argument**

*string*

Constraint argument.

**__str__**

Represent the Constraint instance as a string.

*Returns* The represented Constraint.

*Return type* string

**op_and** (*elements*)

Create an Expression using this Constraint and the specified additional elements joined using an “AND” Operator

*Parameters* *elements* (BaseExpression) – The Expression and/or Constraint elements which the “AND” Operator applies to in addition to this Constraint.

*Returns* Newly created Expression including this Constraint, the elements passed in, and the “AND” Operator.

*Return type* Expression

**op_or** (*elements*)

Create an Expression using this Constraint and the specified additional elements joined using an “OR” Operator

*Parameters* *elements* (BaseExpression) – The Expression and/or Constraint elements which the “OR” Operator applies to in addition to this Constraint.

*Returns* Newly created Expression including this Constraint, the elements passed in, and the “OR” Operator.

*Return type* Expression

**to_python**

Deconstruct the Constraint instance to a tuple.

*Returns* The deconstructed Constraint.

*Return type* tuple

### 5.1.5 Expression

It would be very difficult to build a FIQL Expressions without taking into account the Expressions part of it. The expression module includes the code used for ensuring that any FIQL Expression created with this package is a valid FIQL Expression.

```python
class fiql_parser.expression.BaseExpression
    Bases: object
```
Both Constraint and Expression classes extend the BaseExpression class. A FIQL Constraint is a simple FIQL Expression. As such, they share certain attributes.

Note: The parent of any child of BaseExpression is always an Expression. This is a bit contrary to what might be expected as an Expression itself is a child class of BaseExpression.

This quark is a side effect of the definition of the FIQL Constraint. A FIQL Constraint can not be contained within another FIQL Constraint as a sub-expression. Both a FIQL Constraint and FIQL Expression can only be sub-expressions of an actual FIQL Expression.

```
parent
    Expression
    The Expression which contains this object.

get_parent ()
    Get the parent Expression for this object.
    Returns The Expression which contains this object.
    Return type Expression
    Raises FiqlObjectException – Parent is None.

set_parent (parent)
    Set parent Expression for this object.
    Parameters parent (Expression) – The Expression which contains this object.
    Raises FiqlObjectException – Parent must be of type Expression.
```

class fiql_parser.expression.Expression
    Bases: fiql_parser.expression.BaseExpression

    The Expression is the largest logical unit of a FIQL Expression. It must, like the Constraint evaluate to True or False. The Expression can both contain and be contained by an Expression. It, unlike the Operator and Constraint, MUST contain specific attributes in order to be valid.

    This class contains the bulk of the logic to ensure that an Expression generated by this code is a valid FIQL Expression.

    Note: This Expression class uses a single Operator to join multiple Constraints. This format has the advantage of working cleanly with many ORMs and being far more easily converted to the more string friendly format of Constraint, Operator, Constraint, etc. than the more string friendly format can be converted to the other.

```
elements
    list
    List of Constraint and Expression elements in this Expression.

operator
    Operator
    The Operator which relates the elements in this Expression.

__str__ ()
    Represent the Expression instance as a string.
    Returns The represented Expression.
```
Return type string

**add_element** (*element*)
Add an element of type Operator, Constraint, or Expression to the Expression.

Parameters *element* – Constraint, Expression, or Operator.

Returns self

Return type Expression

Raises FiqlObjectException – Element is not a valid type.

**add_operator** (*operator*)
Add an Operator to the Expression.

The Operator may result in a new Expression if an Operator already exists and is of a different precedence.

There are three possibilities when adding an Operator to an Expression depending on whether or not an Operator already exists:

•No Operator on the working Expression; Simply set the Operator and return self.

•Operator already exists and is higher in precedence; The Operator and last Constraint belong in a sub-expression of the working Expression.

•Operator already exists and is lower in precedence; The Operator belongs to the parent of the working Expression whether one currently exists or not. To remain in the context of the top Expression, this method will return the parent here rather than self.

Parameters *operator* (Operator) – What we are adding.

Returns self or related Expression.

Return type Expression

Raises FiqlObjectExpression – Operator is not a valid Operator.

**create_nested_expression**()
Create a nested Expression, add it as an element to this Expression, and return it.

Returns The newly created nested Expression.

Return type Expression

**has_constraint**()
Return whether or not the working Expression has any Constraints.

Returns Number of logical elements within this Expression.

Return type integer

**op_and** (*elements*)
Update the Expression by joining the specified additional elements using an “AND” Operator

Parameters *elements* (BaseExpression) – The Expression and/or Constraint elements which the “AND” Operator applies to.

Returns self or related Expression.

Return type Expression

**op_or** (*elements*)
Update the Expression by joining the specified additional elements using an “OR” Operator
Parameters `elements (BaseExpression)` – The Expression and/or Constraint elements which the “OR” Operator applies to.

Returns `self or related Expression`.

Return type `Expression`

to_python() – Deconstruct the Expression instance to a list or tuple (If Expression contains only one Constraint).

Returns The deconstructed Expression.

Return type `list or tuple`

5.1.6 Parser

The parser module includes the code used to convert a string representing a FIQL Expression into an object representing the same FIQL Expression.

The Expression object returned is ideally suited for use in filtering database queries with many ORMs.

```python
fiql_parser.parser.iter_parse(fiql_str)
```

Iterate through the FIQL string. Yield a tuple containing the following FIQL components for each iteration:

- preamble: Any operator or opening/closing parenthesis preceding a constraint or at the very end of the FIQL string.
- selector: The selector portion of a FIQL constraint or None if yielding the last portion of the string.
- comparison: The comparison portion of a FIQL constraint or None if yielding the last portion of the string.
- argument: The argument portion of a FIQL constraint or None if yielding the last portion of the string.

For usage see `parse_str_to_expression()`.

```python
Parameters fiql_str (string) – The FIQL formatted string we want to parse.

Yields tuple – Preamble, selector, comparison, argument.
```

```python
fiql_parser.parser.parse_str_to_expression(fiql_str)
```

Parse a FIQL formatted string into an Expression.

```python
Parameters fiql_str (string) – The FIQL formatted string we want to parse.

Returns An Expression object representing the parsed FIQL string.

Return type `Expression`

Raises `FiqlFormatException` – Unable to parse string due to incorrect formatting.
```

Example

```python
>>> expression = parse_str_to_expression(
...   "name==bar,dob=gt=1990-01-01")
```
Python Module Index

f

fiql_parser, 11
fiql_parser.constants, 11
fiql_parser.constraint, 13
fiql_parser.exceptions, 12
fiql_parser.expression, 14
fiql_parser.operator, 12
fiql_parser.parser, 17
### Index

#### Symbols

<table>
<thead>
<tr>
<th>Method/Attribute</th>
<th>Module/Class/Component</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cmp</strong>()</td>
<td>fiql_parser.operator.Operator method</td>
<td>13</td>
</tr>
<tr>
<td><strong>eq</strong>()</td>
<td>fiql_parser.operator.Operator method</td>
<td>13</td>
</tr>
<tr>
<td><strong>lt</strong>()</td>
<td>fiql_parser.operator.Operator method</td>
<td>13</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>fiql_parser.constraint.Constraint method</td>
<td>14</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>fiql_parser.expression.Expression method</td>
<td>15</td>
</tr>
<tr>
<td><strong>str</strong>()</td>
<td>fiql_parser.operator.Operator method</td>
<td>13</td>
</tr>
<tr>
<td>add_element()</td>
<td>fiql_parser.expression.Expression method</td>
<td>16</td>
</tr>
<tr>
<td>add_operator()</td>
<td>fiql_parser.expression.Expression method</td>
<td>16</td>
</tr>
<tr>
<td>ARG_CHAR_REGEX</td>
<td>in module fiql_parser.constants</td>
<td>11</td>
</tr>
<tr>
<td>argument</td>
<td>fiql_parser.constraint.Constraint attribute</td>
<td>14</td>
</tr>
<tr>
<td>ARGUMENT_REGEX</td>
<td>in module fiql_parser.constants</td>
<td>12</td>
</tr>
<tr>
<td>BaseExpression</td>
<td>class in fiql_parser.expression</td>
<td>14</td>
</tr>
<tr>
<td>comparison</td>
<td>fiql_parser.constraint.Constraint attribute</td>
<td>14</td>
</tr>
<tr>
<td>COMPARISON_COMP</td>
<td>in module fiql_parser.constants</td>
<td>12</td>
</tr>
<tr>
<td>COMPARISON_MAP</td>
<td>in module fiql_parser.constraint</td>
<td>13</td>
</tr>
<tr>
<td>COMPARISON_REGEX</td>
<td>in module fiql_parser.constants</td>
<td>11</td>
</tr>
<tr>
<td>Constraint</td>
<td>class in fiql_parser.constraint</td>
<td>13</td>
</tr>
<tr>
<td>CONSTRAINT_COMP</td>
<td>in module fiql_parser.constants</td>
<td>12</td>
</tr>
<tr>
<td>CONSTRAINT_REGEX</td>
<td>in module fiql_parser.constants</td>
<td>12</td>
</tr>
<tr>
<td>create_nested_expression()</td>
<td>(fiql_parser.expression.Expression method)</td>
<td>16</td>
</tr>
<tr>
<td>elements</td>
<td>(fiql_parser.expression.Expression attribute)</td>
<td>15</td>
</tr>
</tbody>
</table>

#### Expression

Expression (class in fiql_parser.expression), 15

#### F

FQL_DELIM_REGEX (in module fiql_parser.constants), 11
fiql_parser (module), 11
fiql_parser.constants (module), 11
fiql_parser.constraint (module), 13
fiql_parser.exceptions (module), 12
fiql_parser.expression (module), 14
fiql_parser.parser (module), 17
FiqlException, 12
FiqlFormatException, 12
FiqlObjectException, 12

#### G

get_parent() (fiql_parser.expression.BaseExpression method), 15

#### H

has_constraint() (fiql_parser.expression.Expression method), 16

#### I

iter_parse() (in module fiql_parser.parser), 17

#### O

op_and() (fiql_parser.constraint.Constraint method), 14
op_and() (fiql_parser.expression.Expression method), 16
op_or() (fiql_parser.constraint.Constraint method), 14
op_or() (fiql_parser.expression.Expression method), 16
Operator (class in fiql_parser.operator), 12
operator (fiql_parser.expression.Expression attribute), 15
OPERATOR_MAP (in module fiql_parser.operator), 12

#### P

parent (fiql_parser.expression.BaseExpression attribute), 15
parse_str_to_expression() (in module fiql_parser.parser), 17

PCT_ENCODING_REGEX (in module fiql_parser.constants), 11

RFC
RFC 3339, 11
RFC 3986#section-2.1, 11
RFC 3986#section-2.3, 11

selector (fiql_parser.constraint.Constraint attribute), 13
SELECTOR_REGEX (in module fiql_parser.constants), 11
set_parent() (fiql_parser.expression.BaseExpression method), 15

T
to_python() (fiql_parser.constraint.Constraint method), 14
to_python() (fiql_parser.expression.Expression method), 17
to_python() (fiql_parser.operator.Operator method), 13

U
UNRESERVED_REGEX (in module fiql_parser.constants), 11

V
value (fiql_parser.operator.Operator attribute), 12